ETH zürich

Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches

1st presented at PACT'12

Gennady Pekhimenko* Vivek Seshadri* Onur Mutlu*

Michael A. Kozuch° Phillip B. Gibbons° Todd C. Mowry*

*Carnegie Mellon University °Intel Labs Pittsburgh

Cliff Hodel

Outline

- Executive Summary
- Background & Problem
- Base + Delta Compression (Base+Δ)
- Base-Delta-Immediate Compression (BΔI)
- Results & Analysis
- Strengths / Weaknesses
- Discussion
- Related Work

Outline

- **Executive Summary**
- Background & Problem
- Base + Delta Compression (Base+Δ)
- Base-Delta-Immediate Compression (BΔI)
- Results & Analysis
- Strengths / Weaknesses
- Discussion
- Related Work

Executive Summary

- **Problem**: Off-chip memory latency is HIGH
  - Increasing cache size has significant drawbacks
  - Apply compression to increase cache capacity
- **Challenge**: Decompression is on critical path
- **Goal**: New compression mechanism
  - 1. low decompression latency, 2. low HW complexity, 3. high compression ratio
- **Key-insight**: 4 data patterns can be combined in one general notion
- **Base-Delta-Immediate (BΔI) Compression**:
  - Key-Idea: Encode cache lines as one base + array of differences to base
  - BΔI outperforms three prior mechanisms in compression ratio and decompression latency

Outline

- Executive Summary
- **Background & Problem**
- Base + Delta Compression (Base+Δ)
- Base-Delta-Immediate Compression (BΔI)
- Results & Analysis
- Strengths / Weaknesses
- Discussion
- Related Work

Background & Problem:

- **Why compression?**
- Patterns to make use of
- 3 prior mechanisms

Why compression?

Cache

- Bigger Cache:
  - More capacity → Fewer cache misses
  - Longer access latency
  - Increased area
  - Higher power consumption

Increasing cache size has too many drawbacks!

Why compression?

Compressed Cache

- Compressed Cache:
  - More capacity → Fewer cache misses
  - Longer access latency
  - Increased area
  - Higher power consumption

but much less than before

Compressed cache increases capacity without major drawbacks!

(if we keep access latency low)

Background & Problem:

- Why compression?
- **Patterns to make use of**
- 3 prior mechanisms

Data Patterns we can make use of

1. Zeros: by far the most seen value in real-world applications:

- initializing data,
- representing NULL-pointers,
- false Booleans,
- representing sparse matrices (in dense form especially).

0x00000000	0x00000000	0x00000000	…	0x00000000

2. Repeated values: a single value repeated many times: e.g:

- common initial value for large arrays
- in multimedia apps: large number of pixels with same colour

0x00C61FF	0x00C61FF	0x00C61FF	…	0x00C61FF

Data Patterns we can make use of

3. Narrow values: a small value stored using large datatype:

- appear commonly because programmers over-provision data type sizes

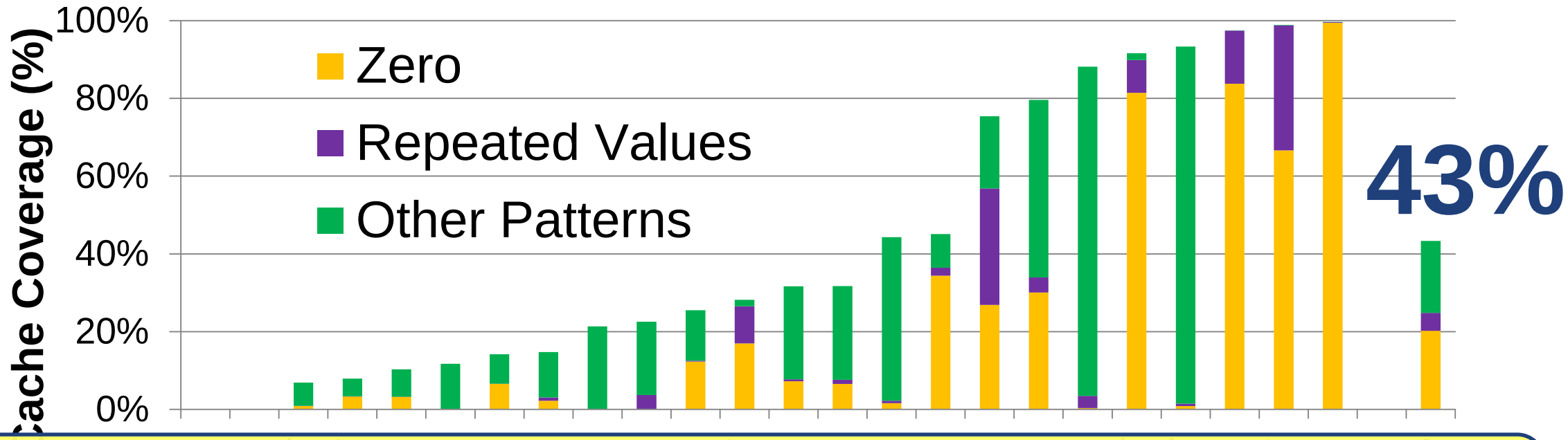
0x00000003	0x00000000	0x00000005	…	0x00000002

4. Low range:

- table of pointers that point to different locations in the same memory region,
- image with low colour gradient

0xC0403E10	0xC0403E18	0xC0403E20	…	0xC0403EE0

Observation



Patterns highly present in many applications!

Background & Problem:

- Why compression?
- Patterns to make use of
- **3 prior mechanisms**

3 prior mechanisms

- Zero-Content Augmented (ZCA) cache
- Frequent Value Cache (FVC)
- Frequent Pattern Compression (FPC)

Zero-Content Augmented (ZCA) cache: 2009

	Zeros	Rep. Values	Narrow Values	Low Range
ZCA	✓	✗	✗	✗

Low compression ratio

Frequent Value Cache (FVC): 2000

	Zeros	Rep. Values	Narrow Values	Low Range
FVC	✓	✓/✗	✗	✗

Too high latency and complexity for modest compression ratio

Frequent Pattern Compression (FPC): 2004

	Zeros	Rep. Values	Narrow Values	Low Range
FPC	✓	✓	✓	✗

Too high decompression latency

3 prior Mechanisms: Summary

	Zeros	Rep. Values	Narrow Values	Low Range
ZCA	✓	✗	✗	✗
FVC	✓	✓/✗	✗	✗
FPC	✓	✓	✓	✗

3 prior Mechanisms: Summary

	Zeros	Rep. Values	Narrow Values	Low Range
ZCA	✓	✗	✗	✗
FVC	✓	✓/✗	✗	✗
FPC	✓	✓	✓	✗
BΔI	✓	✓	✓	✓

Outline

- Executive Summary
- Background & Problem
- **Base + Delta Compression (Base+Δ)**
- Base-Delta-Immediate Compression (BΔI)
- Results & Analysis
- Strengths / Weaknesses
- Discussion
- Related Work

Base + Delta Compression (Base+Δ)

- **Basic Idea**
- Compression
- Decompression
- Changes to Cache

Key-Observation of BΔI-Paper

- 1. Zeros
- 2. Repeated Values
- 3. Narrow Values
- 4. Other Patterns

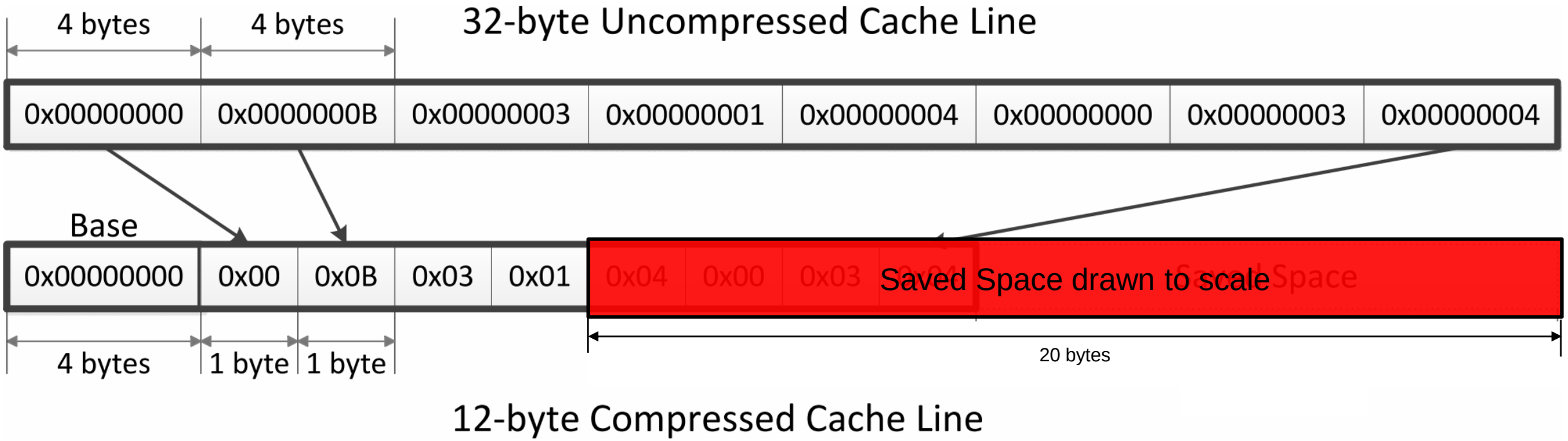
Low Dynamic Range

Base + Delta compression: Basic Idea

- Compression at cache line granularity
  - Compress whole cache line or store complete uncompressed line
- Compress line as: Base + array of differences

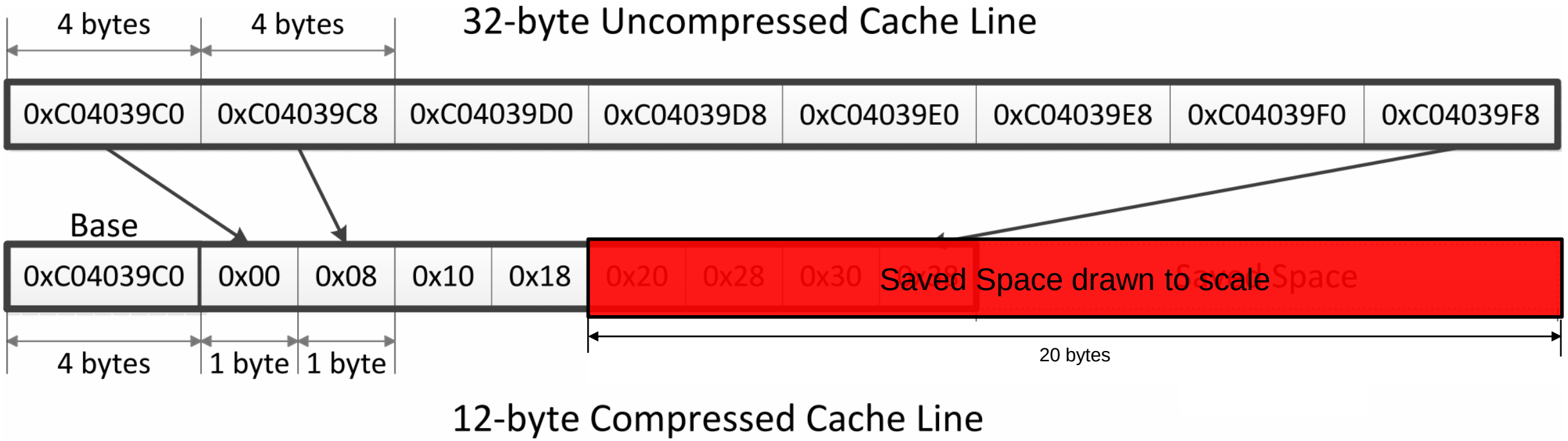
Two basic examples (32-byte cache lines)

- Narrow values stored in 4-byte ints (from application *h264ref*):



Two basic examples (32-byte cache lines)

- Nearby pointers stored in same cache line (from application *perlbench*):

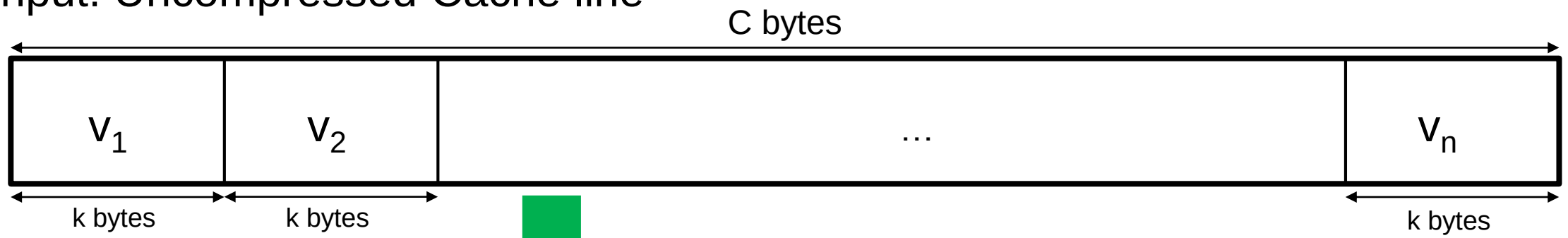


Base + Delta (Base + Δ) Compression

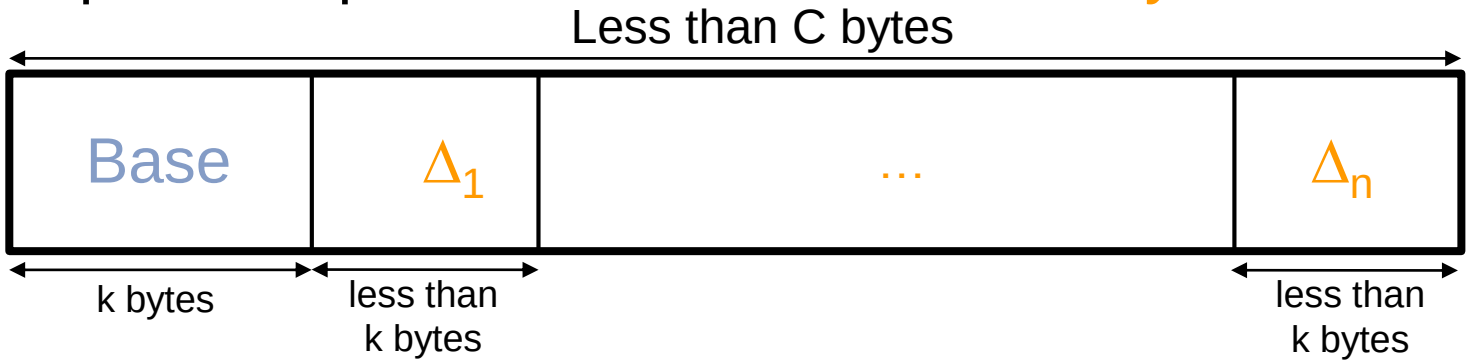
- Basic Idea
- **Compression**
- Decompression
- Changes to Cache

Compression Algorithm

- Input: Uncompressed Cache line



- Output: Compressed Line as Base + Array of deltas



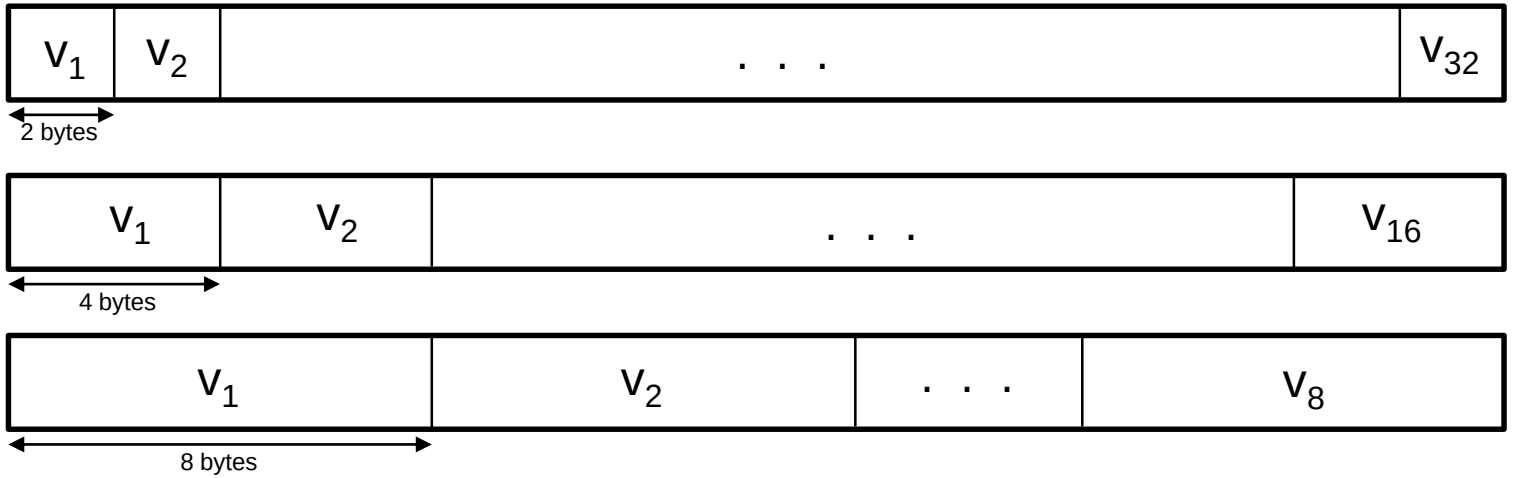
Two observations

- **1)** cache line compressible $\Leftrightarrow$ $\forall i \; size(\Delta_i) < k$

- **2)** optimal values for Base:
  - Minimum or maximum of all values in cache line
  - Or exactly in between

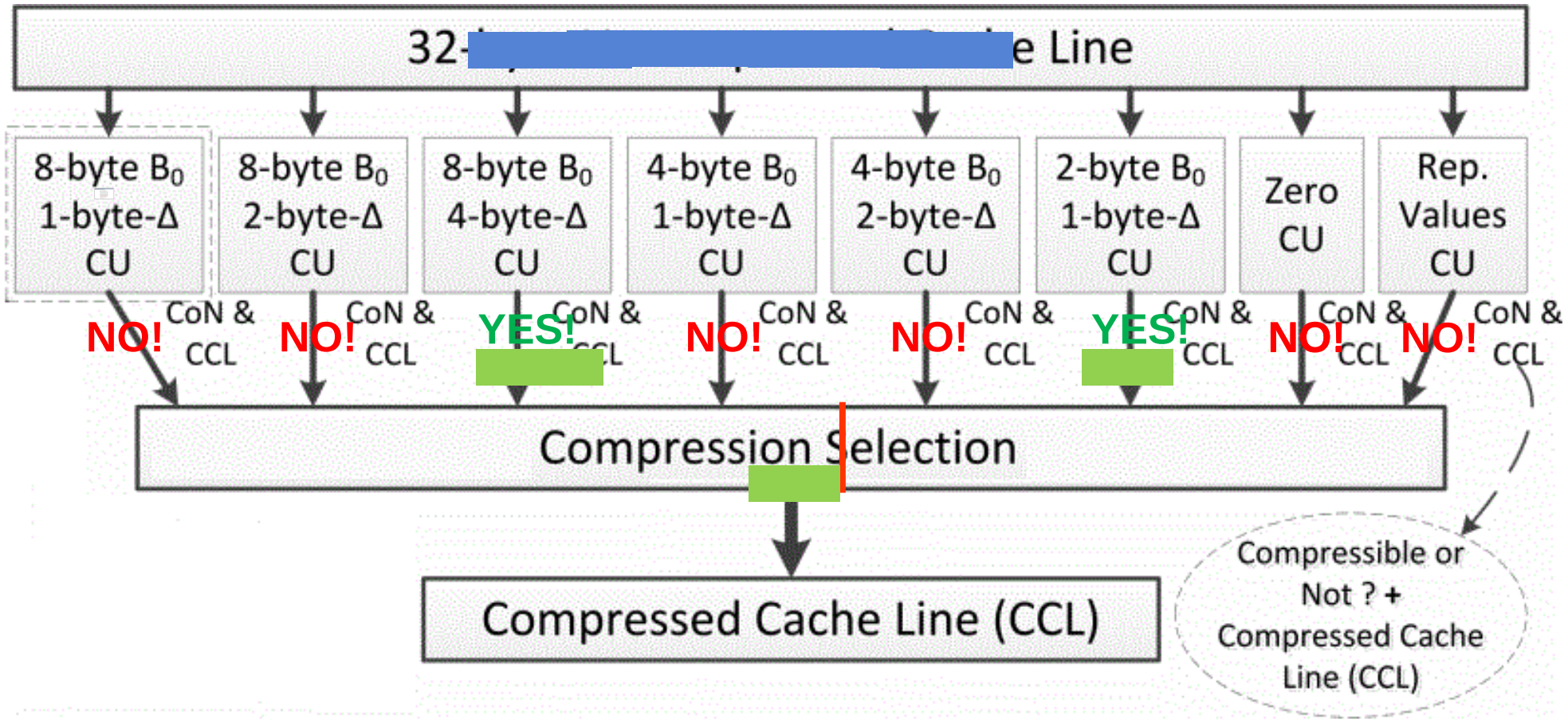
Determining k and the Base

- k:
  - We consider k ∈ {2, 4, 8}

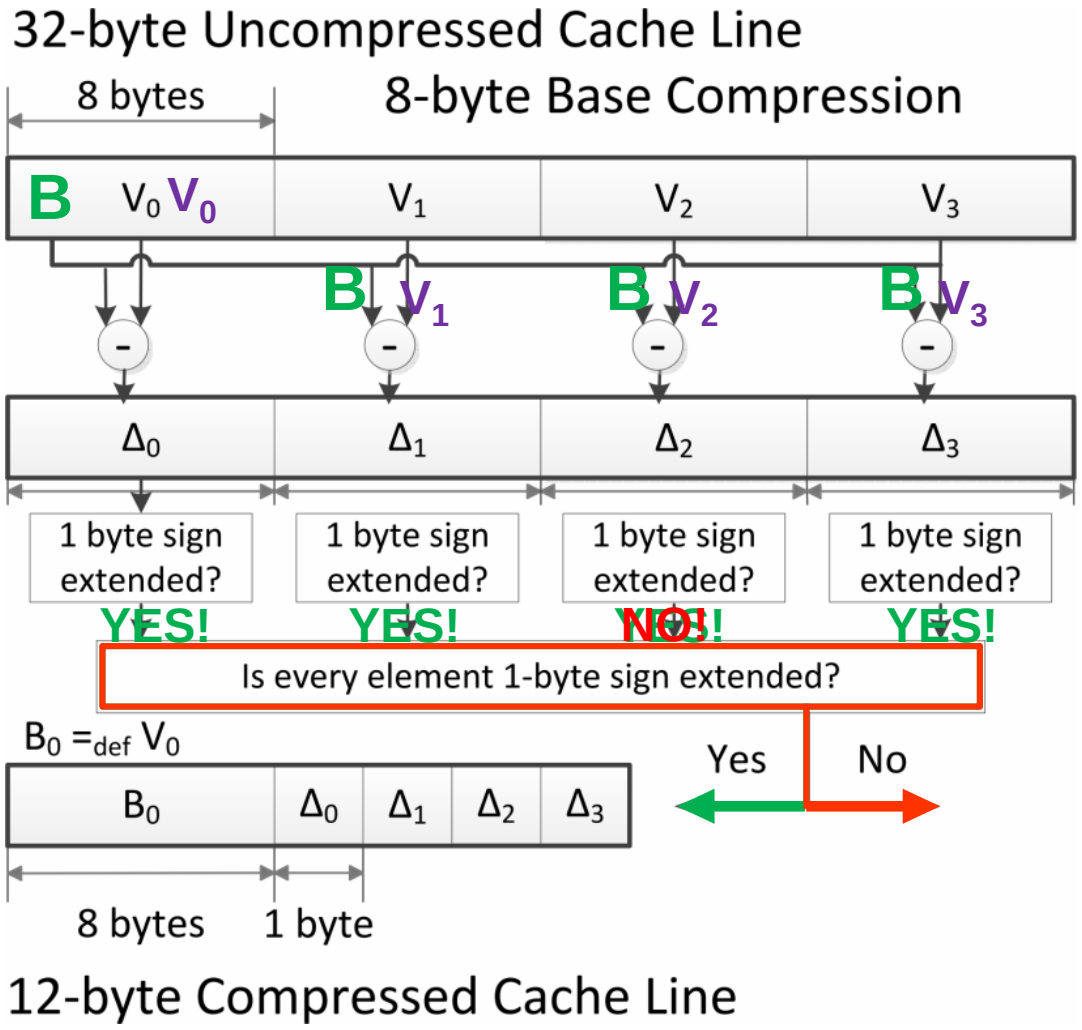


  - We choose the k which provides the most compression
- Base:
  - We simply choose Base as the 1st value in line
  - Choosing first value as Base instead of optimal Base decreases average compression ratio only by 0.4%

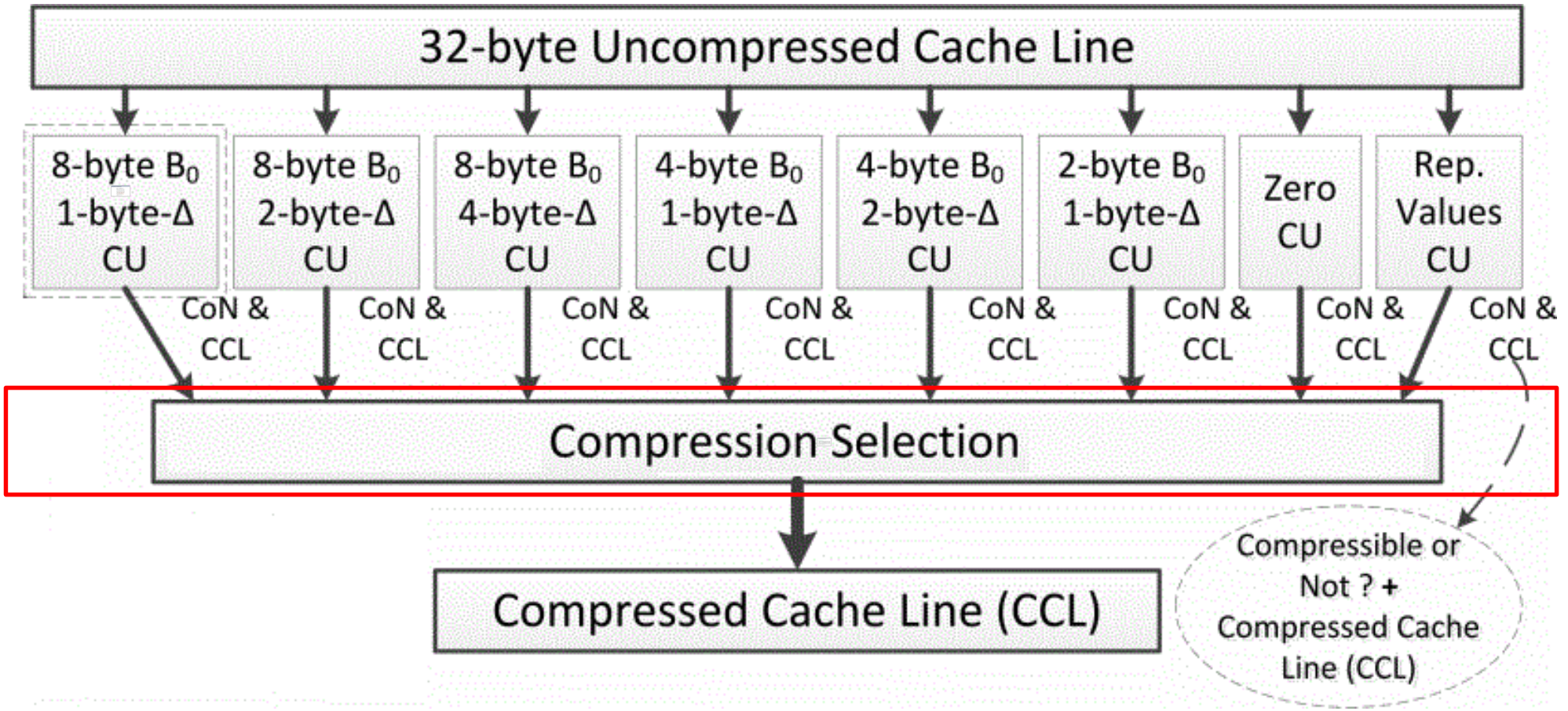
BΔI Compression: Overview



BΔI Compression: Compressor Unit



BΔI Compression: Overview

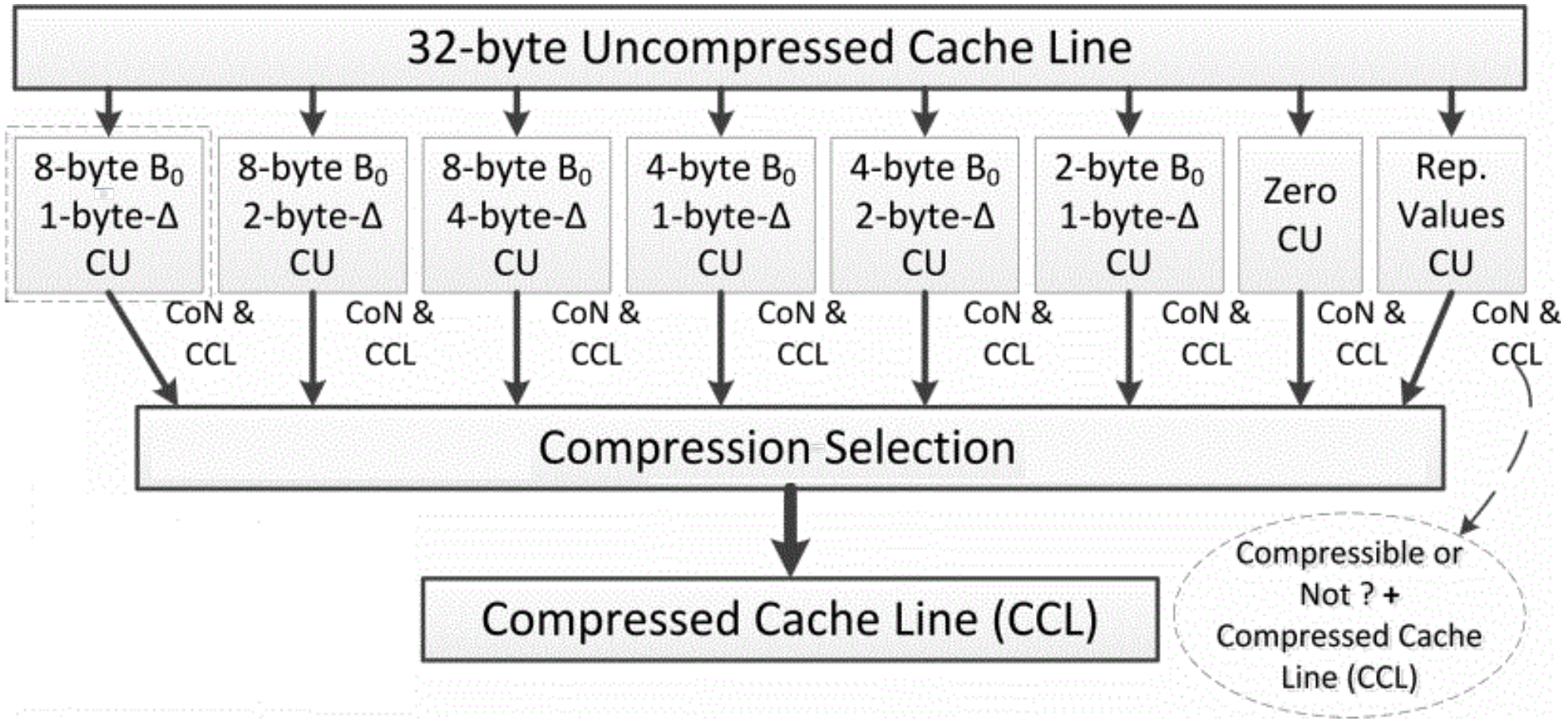


B∆I Compression: Compression Selection

- **Sizes are statically known!**

Name	Base	Δ	Size	Enc.	Name	Base	Δ	Size	Enc.
Zeros	1	0	1/1	0000	Rep.Values	8	0	8/8	0001
Base8-Δ1	8	1	12/16	0010	Base8-Δ2	8	2	16/24	0011
Base8-Δ4	8	4	24/40	0100	Base4-Δ1	4	1	12/20	0101
Base4-Δ2	4	2	20/36	0110	Base2-Δ1	2	1	18/34	0111
NoCompr.	N/A	N/A	32/64	1111					

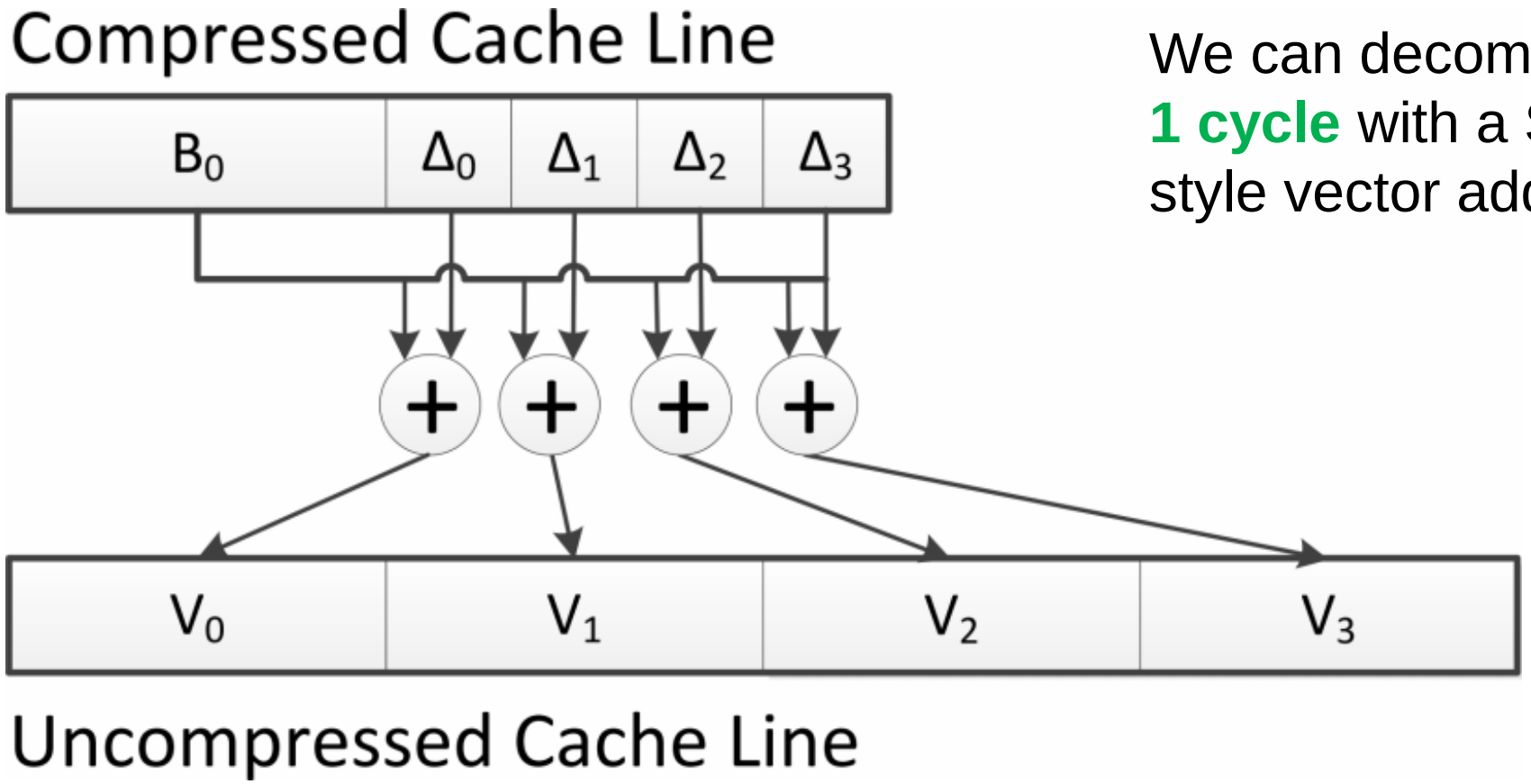
BΔI Compression: Overview



Base + Delta (Base + Δ) Compression

- Basic Idea
- Compression
- **Decompression**
- Changes to Cache

BΔI Decompression Logic



We can decompress in **1 cycle** with a SIMD style vector addition

Base + Delta (Base + Δ) Compression

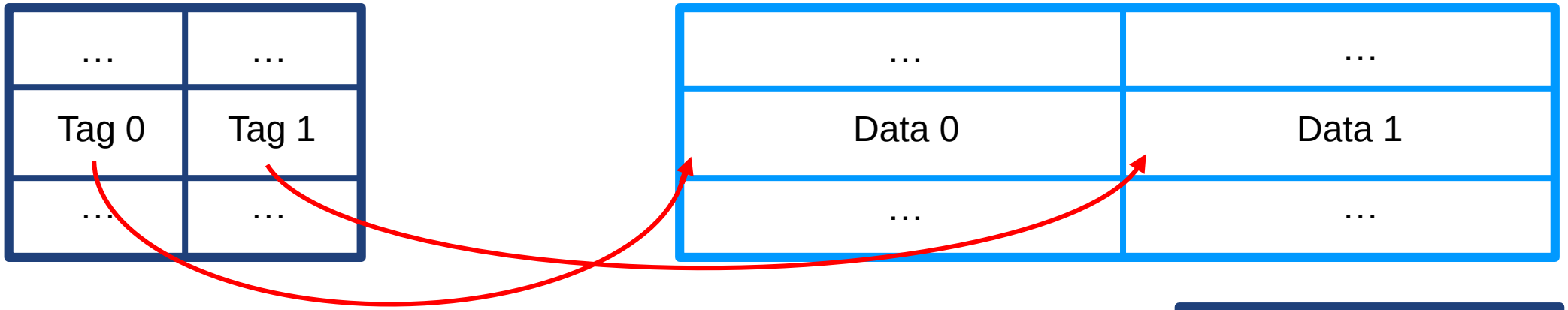
- Basic Idea
- Compression
- Decompression
- **Changes to Cache**

Cache Organization: Encoding bits

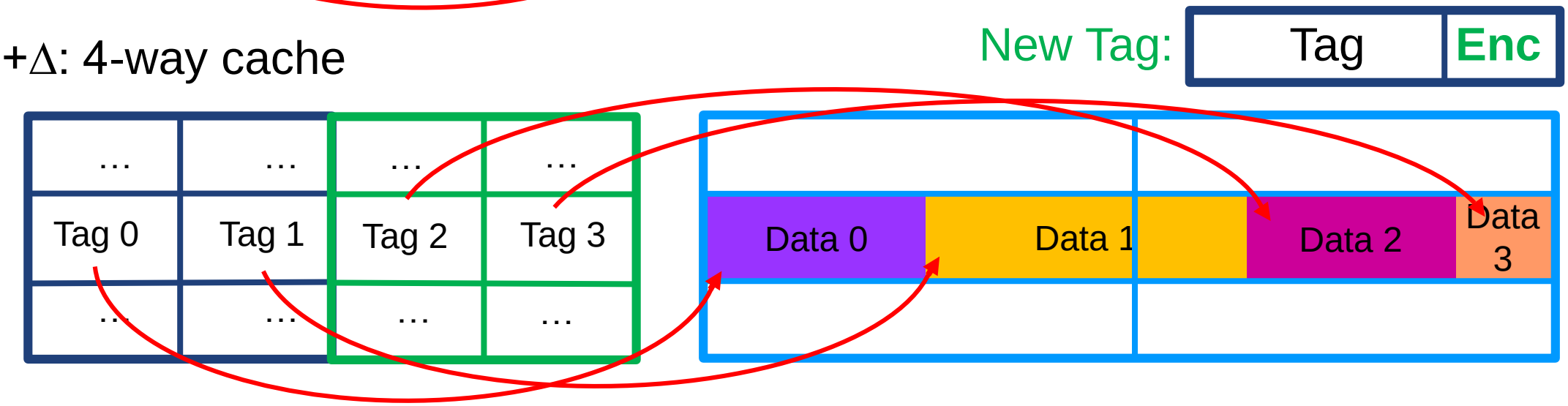
Name	Base	Δ	Size	Enc.	Name	Base	Δ	Size	Enc.
Zeros	1	0	1/1	0000	Rep.Values	8	0	8/8	0001
Base8-Δ1	8	1	12/16	0010	Base8-Δ2	8	2	16/24	0011
Base8-Δ4	8	4	24/40	0100	Base4-Δ1	4	1	12/20	0101
Base4-Δ2	4	2	20/36	0110	Base2-Δ1	2	1	18/34	0111
NoCompr.	N/A	N/A	32/64	1111					

BΔI Cache Organization

- Conventional: 2-way cache, 32-byte cache lines



- B+Δ: 4-way cache



Outline

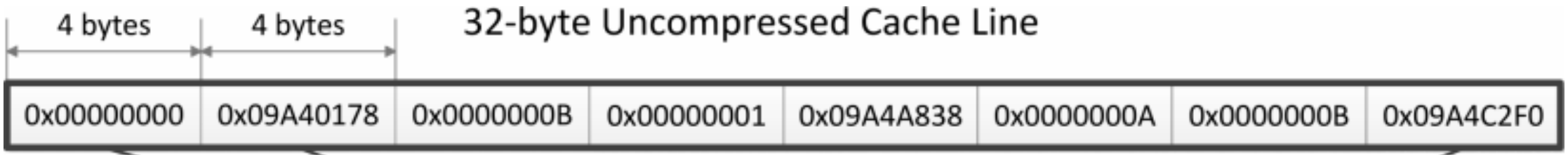
- Executive Summary
- Background & Problem
- Base + Delta Compression (Base+Δ)
- **Base-Delta-Immediate Compression (BΔI)**
- Results & Analysis
- Strengths / Weaknesses
- Discussion
- Related Work

Base-Delta-Immediate Compression (BΔI)

- **Base+Δ Limitations**
- Multiple Bases
- From Base+Δ to BΔI
- BΔI Storage Costs
- Eviction Policy

Base+Δ Limitations

- Example cache line from *mcf:*



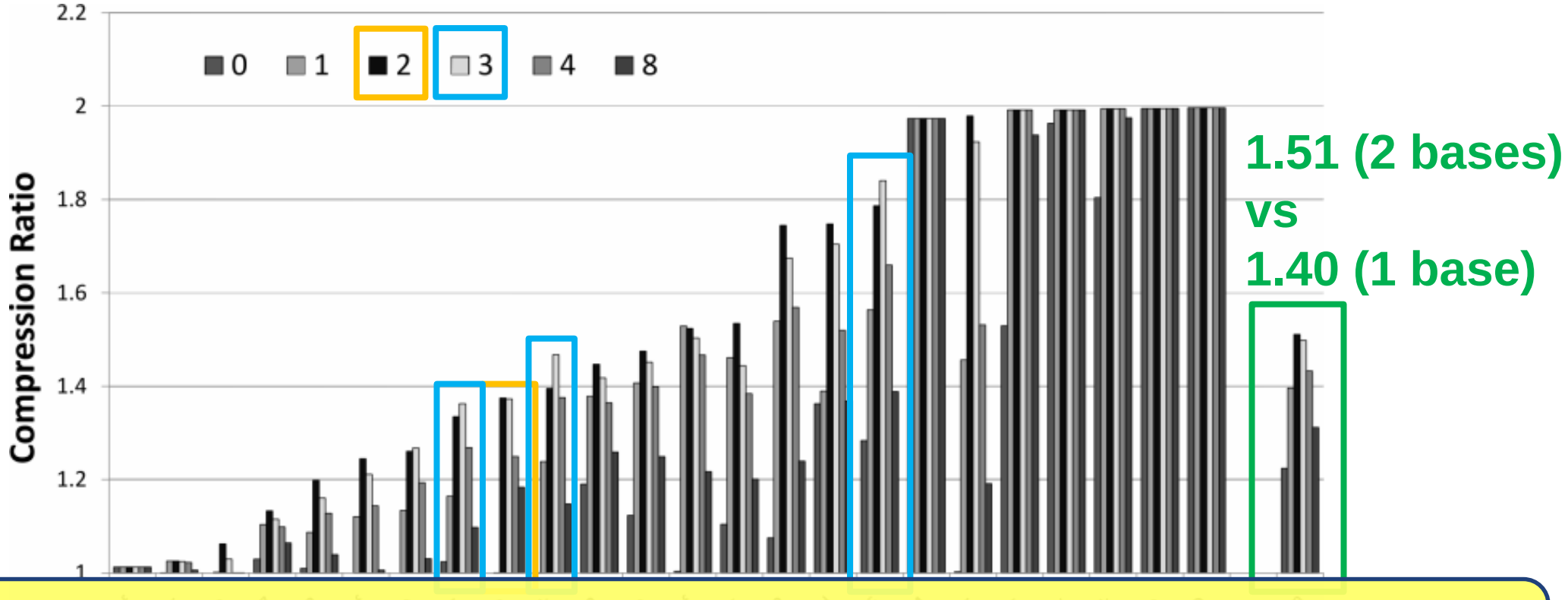
- Not compressible with current mechanism
- Compressible with two bases

Base-Delta-Immediate Compression (BΔI)

- Base+Δ Limitations
- **Multiple Bases**
- From Base+Δ to BΔI
- BΔI Storage Costs
- Eviction Policy

Even more bases?

0 bases = only compress simple patterns like zeros and repeated values



2 bases best on average

Problem: Finding 2nd base

- Idea: Choose 2nd base to be implicitly 0
- Why could this work?
  - Aggregate data types, e.g. structs in C
  - E.g. pointers mixed with small integers

- Deltas to 0 can be seen as immediate values

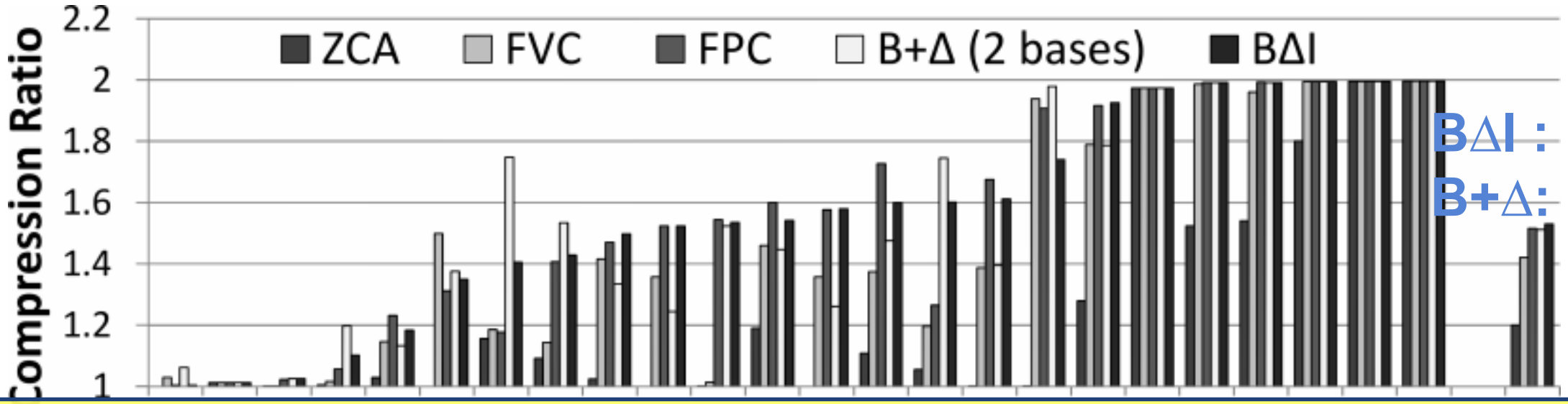
  → Base-Delta-Immediate compression

BΔI vs Base+Δ with 2 arbitrary bases

- BΔI:
  - Implicit 2nd base → less storage overhead
    → potentially higher compression ratio
- Base+Δ with two arbitrary bases:
  - More storage to store arbitrary 2nd base value
  - Can compress more cache lines
    → potentially higher compression ratio

→ Compare them on the benchmarks

Compression ratio comparison of different algorithms



BΔI and Base+Δ (2 bases) work well.
BΔI simpler!

Base-Delta-Immediate (BΔI) Compression

- Base+Δ Limitations
- Multiple Bases
- **From Base+Δ to BΔI**
- BΔI Storage Costs
- Eviction Policy

From B+Δ to BΔI

- Compression in 2 steps:
  - Step 1: Try to compress all elements using implicit base of 0
  - Step 2: Compress elements which weren't compressed in step 1 with arbitrary base
  - Base: First element, which wasn't compressed in step 1
  - Stores a bit mask, 1-bit per element:
    - → immediate value
    - → difference to base
- Decompression:
  - Masked addition of the base to the array of differences, using bit mask

B∆I Operation

- At cache levels higher than L1
- Latency at level 1 too important, but in principle would be possible

Base-Delta-Immediate Compression (BΔI)

- Base+Δ Limitations
- Multiple Bases
- From Base+Δ to BΔI
- **BΔI Storage Costs**
- Eviction Policy

B∆I Storage Cost: Changes

- 2MB 16-way L2 cache, assuming 64-byte cache lines:

	Baseline	B∆I
Size of tag-store entry	21 bits	32 bits (+4–encoding, +7–segment pointer)
Size of data-store entry	512 bits	512 bits
Number of tag-store entries	32768	65536
Number of data-store entries	32768	32768
Tag-store size	84kB	256kB
Total (data-store+tag-store) size	2132kB	2294kB

Adding 11 bits

Doubling number of tag-store entries

Increase by a factor of ~3

Overall: Modest increase (1.07x)

Base-Delta-Immediate Compression (BΔI)

- Base+Δ Limitations
- Multiple Bases
- From Base+Δ to BΔI
- BΔI Storage Costs
- **Eviction Policy**

Eviction Policy

- Problem: Evicting one cache line may not create enough space for incoming cache line
- 5.2% of all insertions/writebacks result in evicting multiple lines on average
- Paper uses policy that evicts multiple LRU cache lines
- But paper leaves it open for future work to get better eviction policy

Outline

- Executive Summary
- Background & Problem
- Base + Delta Compression (Base+Δ)
- Base-Delta-Immediate Compression (BΔI)
- **Results & Analysis**
- Strengths / Weaknesses
- Discussion
- Related Work

Results and Analysis

- **Evaluation Methodology**
- Single-Core results
- Multi-Core results

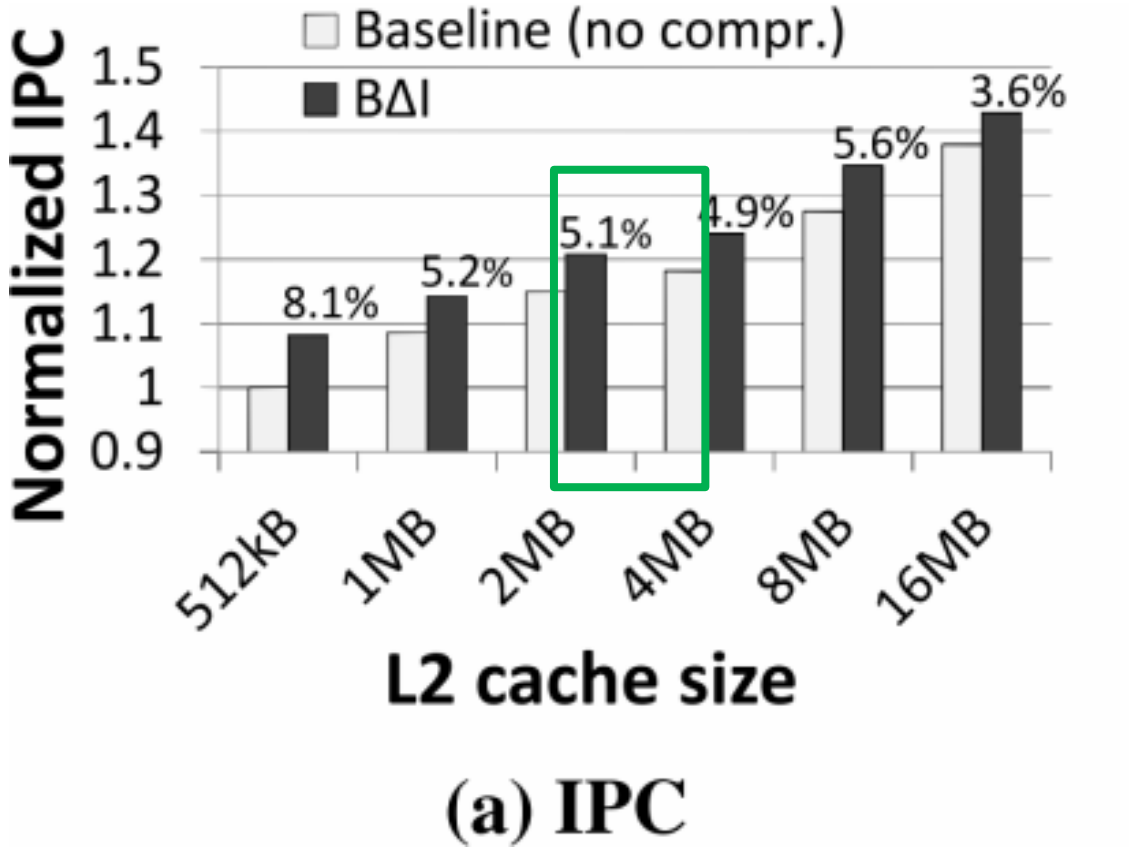
Evaluation Methodology

- On a simulator:
  - 2 or 3 level hierarchy
  - 64-byte cache lines
- B∆I caches have:
  - +1 / +2 cycles latency on hit/miss (larger tag store)
  - +1 cycle latency due to decompression
  - Therefore +2 / +3 cycles added latency
- Assumption: No internal fragmentation

Results and Analysis

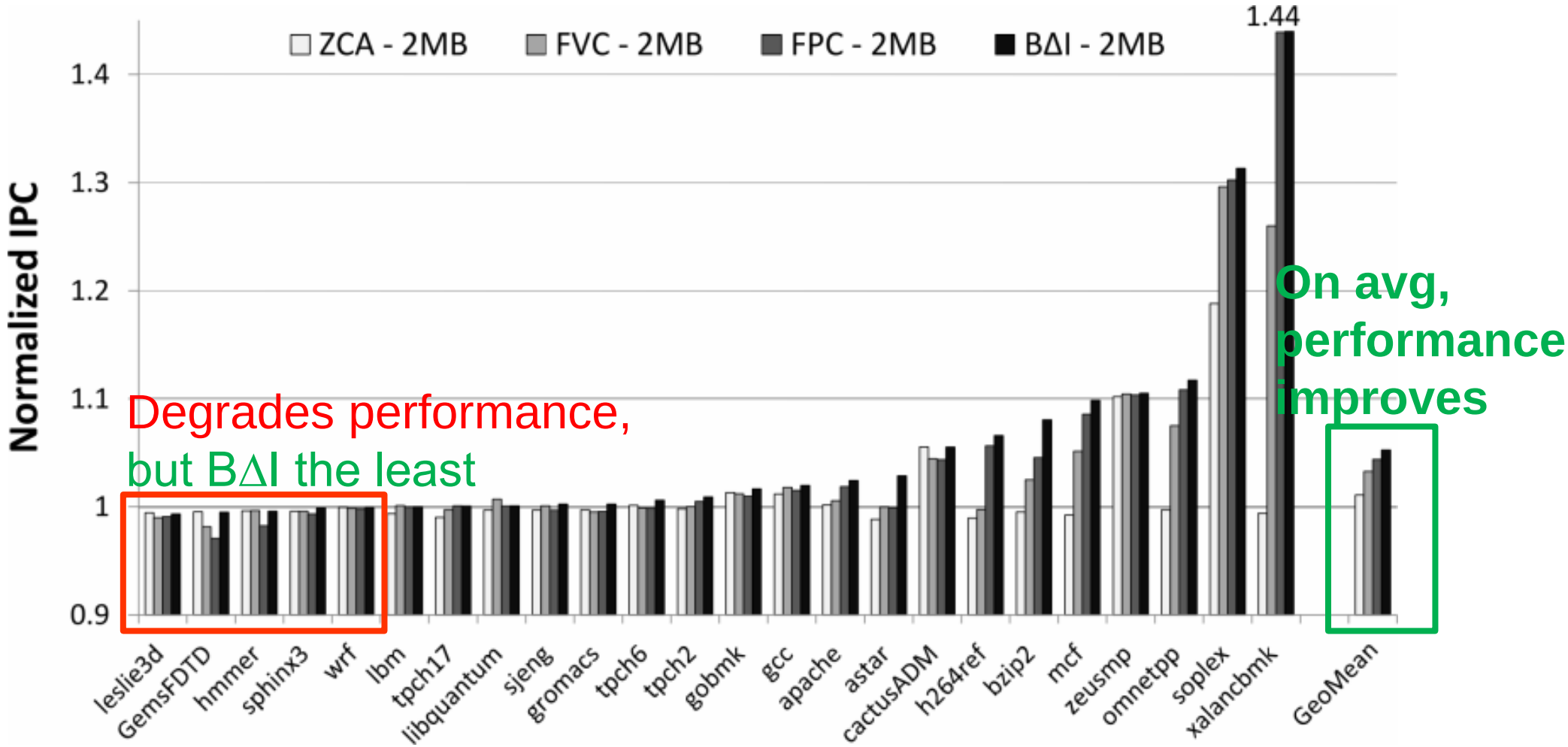
- Evaluation Methodology
- **Single-Core results**
- Multi-Core results

Single-Core results



(a) IPC

Single-Core results



Results and Analysis

- Evaluation Methodology
- Single-Core results
- **Multi-Core results**

Multi-Core results

- Interesting: Shared L2, L3
- BΔI gives perfomance improvement over all prior mechanisms:

Cores	No Compression	ZCA	FVC	FPC
1	5.1%	4.1%	2.1%	1.0%
2	9.5%	5.7%	3.1%	1.2%
4	11.2%	5.6%	3.2%	1.3%

Outline

- Executive Summary
- Background & Problem
- Base + Delta Compression (Base+Δ)
- Base-Delta-Immediate Compression (BΔI)
- Results & Analysis
- **Strengths / Weaknesses**
- Discussion
- Related Work

Strengths

- Elegant and intuitive idea: One model fits all
- Evaluation quite thorough:
  - Many things tested:
    - # of bases
    - # of tags
  - Fair comparison to prior mechanisms
  - Especially Multi-core analysis
- Individual parts well-explained

Weaknesses

- Paper leaves out some things:
  - Replacement policy
  - Internal Fragmentation
- Transition from Base+Δ to BΔI really short
- Paper jumps around a lot

Outline

- Executive Summary
- Background & Problem
- Base + Delta Compression (Base+Δ)
- Base-Delta-Immediate Compression (BΔI)
- Results & Analysis
- Strengths / Weaknesses
- **Discussion**
- Related Work

Discussion

- Any ideas for improvements to BΔI?
- Ideas for other cache compression mechanisms?

Related Work

Doppelgänger: A Cache for Approximate Computing

Joshua San Miguel
University of Toronto
joshua.sanmiguel@mail.utoronto.ca

Jorge Albericio
University of Toronto
jorge@eecg.toronto.edu

Andreas Moshovos
University of Toronto
moshovos@eecg.toronto.edu

Natalie Enright Jerger
University of Toronto
enright@ece.utoronto.ca

Touché: Towards Ideal and Efficient Cache Compression By Mitigating Tag Area Overheads

Seokin Hong*, Bulent Abali†, Alper Buyuktosunoglu†, Michael B. Healy†, and Prashant J. Nair‡

*Kyungpook National University
seokin@knu.ac.kr

†IBM T. J. Watson Research Center
[*abali,alperb,mbhealy*]*@us.ibm.com*

‡The University of British Columbia
prashantnair@ece.ubc.ca

Discussion

- Any ideas for improvements to BΔI?
- Ideas for other cache compression mechanisms?
- Is cache compression actually used in today's processors?
  - Why not?
  - Frame Buffer Compression: reduce memory bandwidth due to display traffic
- Does BΔI bring any security risks with it?
- Will cache compression get more important in the future?

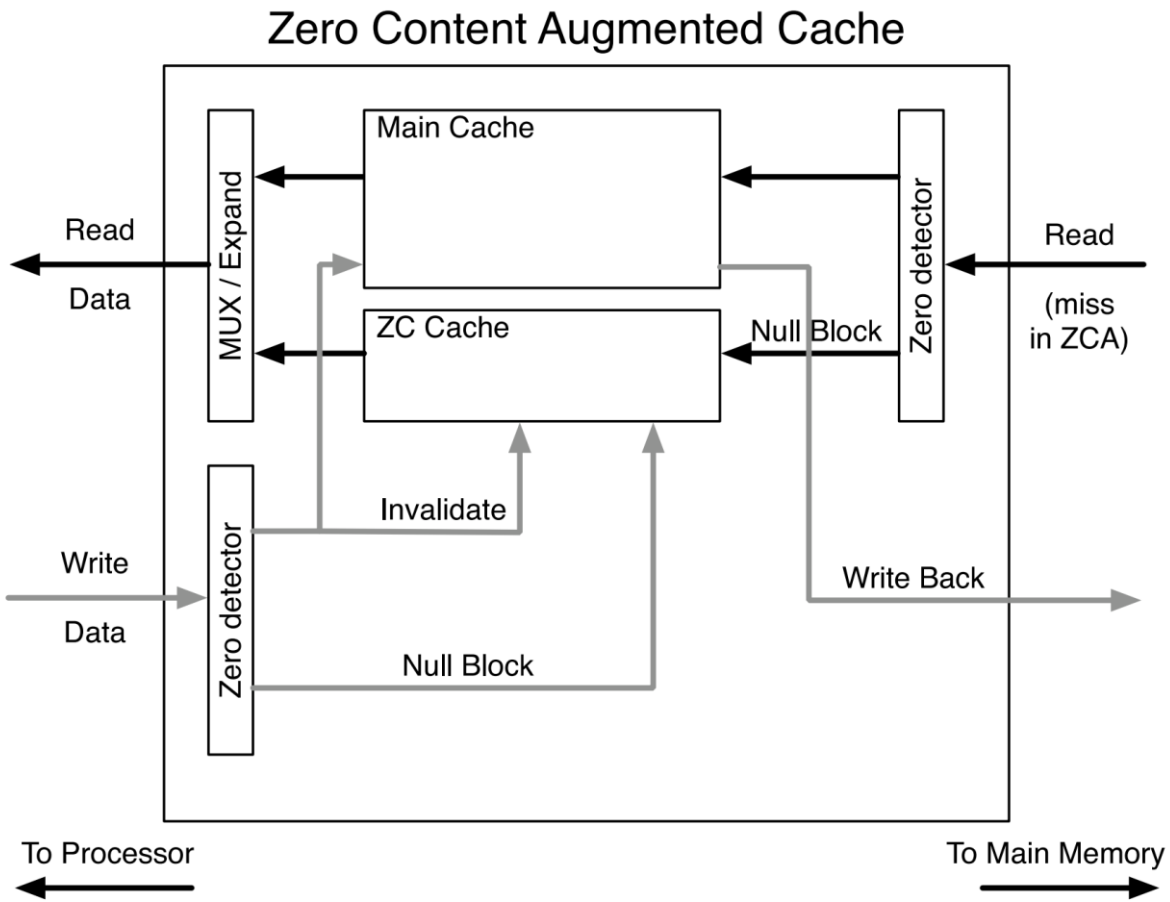
Slides not shown at presentation:

Zero-Content Augmented (ZCA) cache: 2009

- Idea: Compress zero memory blocks using a specialized cache
  - called “ZC cache”
- One zero cache line represented solely by its address tag and a single valid bit
  - Optimization: ZC cache entry consists of shortened address tag and N valid bits

    → single entry in ZC cache can map up to N null lines
- “Compression”: OR-ing whole line to check if all zero

ZCA cache: conventional chache + ZC cache



Frequent Value Cache (FVC): 2000

- Observation: *Frequent value locality*: 10 distinct values occupy over 50% of all memory locations and account on average nearly 50% of all memory accesses
- Idea: Use a Frequent Value Cache (FVC) along with a traditional cache
- FVC stores lines via 3-bit encoding

  → 7 most frequent values, plus last code for “infrequent value”

B∆I Compression

- Compression logic consists of 8 distinct compressor units:
  - 6 for different base sizes and ∆ sizes
  - 2 for zeros and repeated value compression
- Uncompressed cache line → Compressor unit → "Not compressible"
  → "Compressible" + compressed cache line
- In the end, choose output from compressor unit, which compresses the most

Zero-Content Augmented (ZCA) cache: 2009

- Basic Idea: One zero cache line represented solely by its address tag and a single valid bit
- Can compress: Zeros

Low compression ratio

Frequent Value Cache (FVC): 2000

Frequent Value (32 Bits)	0	-1	1	2	4	8		Infrequent values
3 Bit Encoding	000	001	010	011	100	101	110	

Too high latency and complexity for modest compression ratio

000	111	000	111	001	110	010	001

FVC: 8 word compressed cache line (24 Bits)

- Can compress: **Zeros** **Repeated Values (sometimes)**

Frequent Pattern Compression (FPC): 2004

Table 1. Frequent Pattern Encoding

Prefix	Pattern Encoded	Data Size
000	Zero Run	3 bits (for runs up to 8 zeros)
001	4 bit sign-extended	4 bits
010	One byte sign-extended	8 bits
011	halfword sign-extended	16 bits
100	halfword padded with a zero halfword	The nonzero halfword (16 bits)
101	Two halfwords, each a byte sign-extended	The two bytes (16 bits)
110	word consisting of repeated bytes	8 bits
111	Uncompressed word	Original Word (32 bits)

Too high decompression latency

- Can compress: **Zeros** **Repeated Values** **Narrow Values**